CS 308 - EMBEDDED SYSTEMS LAB

Final Project Report

TEAM NAME: HEROBOT!
TEAM CODE: TU-4

MEMBERS:
Suraj Sachdev (100050005)
Praveen Sampath (100050004)
Krishna Pillutla (100050067)
Amit Panghal (100050016)

April 18, 2014

Contents

1 Introduction **3**

2 Problem Statement **3**

3 Requirements **4**
3.1 Functional Requirments 4
3.2 Non-functional Requirments 4
3.3 Hardware Requirments 4
3.4 Software Requirments 5

4 System Design **5**

5 Working of System and Test Results **8**

6 Discussion of System **9**

7 Future Work **10**

8 Conclusion **11**

9 References **11**

1 Introduction

This report discusses herobot, a Firebird V robot coupled with a robotic arm controlled by hand gestures. The robotic arm has a gripper that can be used to pick and place small objects around the mobile Firebird V robot. The user is presented with an interface that captures palm and finger movements to control the robotic arm.

The existing interfaces for robotic arms are not very intuitive. Therefore, we decided to provide the user with a very intuitive hand gesture based interface that provides a very fine-grained control of the robotic arm. Motions of the user's palm and fingers are mapped to motions of the robotic arm and the gripper.

Such a system has several applications, especially in environments where human presence is not feasible. Coupled with a robotic arm that is controlled via an intuitive gesture-based interface, herobot can perform tasks that require the dexterity of the human hand. One example of a scenario where such a system would be useful is that of an automated greenhouse. The person controlling the greenhouse remotely can also control the bot to perform tasks such as harvesting fruits, moving things around, etc.

2 Problem Statement

In this project, we aimed to built a robotic arm that can be remotely controlled and used as a substitute to the human hand at places where human presence is not feasible. We planned to achieve following in our project:

(1) Devise intuitive hand gestures to control robotic arm

(2) Using a Leap Motion controller to capture hand gestures and process these gestures using the Leap Motion API, for sending appropriate commands to the bot

(3) Process the commands sent from remote user on the bot and actuate appropriate servo motors for desired movement of robotic arm and bot

(4) Get a live feedback of current surroundings of the bot through camera mounted on bot for better handling

Our bot can serve as basic building block of much larger applications where fine-tuned remote control of a robotic arm is required. Simple examples are fruit-plucking bot with other advanced features like voice control, etc.

3 Requirements

3.1 Functional Requirments

In this section, we reproduce the functional requirements we had described in the Software Requirements Specification:

The Firebird V bot has a robotic arm mounted on it. The user controls the both the movement of the bot and the robotic arm. The user controls the motion of the bot through keys on an android app (or a computer's keyboard) and the motion of the robotic arm through a leap motion device. The bot will host an Android device to provide video feed to the user. This device will also be used to communicate the bots movement control signals from the signals using a Bluetooth module attached to the bot. The user can view the video remotely to view the surroundings of the bot. The robotic arm is finally to be used for purposes such as plucking tomatoes in the greenhouse.

Movement of bot
The bot movement is controlled via a user interface on an Android phone. Our system extends it to a simple Java application for a PC. The user uses the keyboard to send control signals to move the bot forward, backward, left and right.

Movement of arm
The arm movement is controlled by hand gestures made by the user. The gestures are made as intuitive as possible, so that the robotic arm can be controlled easily. The user is able to raise or lower the arm to reach objects, and grasp them using the gripper - all using simple hand gestures.

3.2 Non-functional Requirments

(a) The movement of the arm is determined by the orientation of the users palm, i.e., if the users palm is horizontal, the arm is bent till the griper is at the level of the base, and when the palm is vertical, the arm is erect

(b) The distance between the grippers in the robotic arm is determined by distance between the thumb and the forefinger of the controlling arm

(c) The controlling arm is the rightmost arm in the field of view of the Leap Motion Controller

(d) Video feed is provided by the same Android phone used for communication with the bot

3.3 Hardware Requirments

The following hardware components were used in our project:

- FireBird V bot with ATmega 2560 processor
- 4-axis robotic arm by Nex Robotics (or an equivalent)
- Leap Motion Controller [3]
- Android Mobile (for video streaming)
- Xbee wireless chip (configured pair)

3.4 Software Requirments

The following software packages/libraries were used in our project:

- AVR Studio [1]
- AVR Bootloader [2]
- Leap Motion Java SDK and drivers [4]
- Java Simple Serial Connector [5]
- Serial Terminal (for debugging)

4 System Design

The Finite State Machine representing our system has four states:

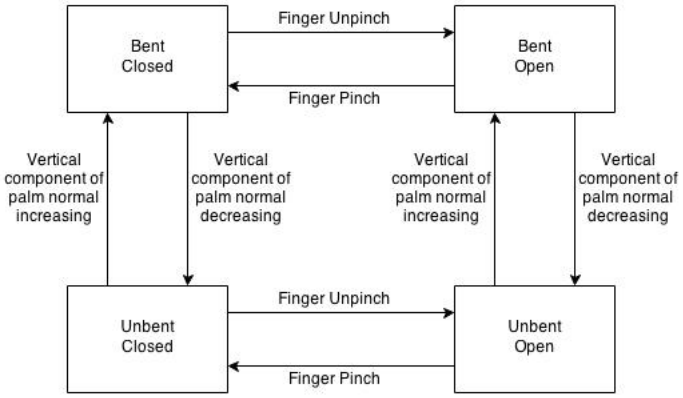

Figure 1: FSM

The (originally planned) architecture of our system is:

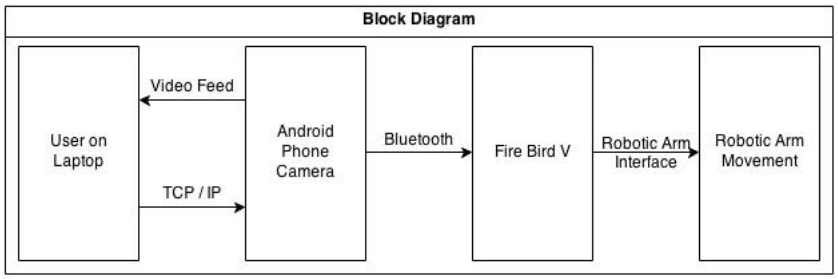

Figure 2: Architecture Block Diagram

Workflow: The flow of messages that occurs in the system when the user makes hand gestures is as follows:

a. The Leap Motion sensor captures frames

b. Java code running on the laptop to which the sensor is connected is running a listener (Java class) which receives these frames

c. The listener extracts useful information from the frames like palm orientation, distance between fingers, etc.

d. The information from the frames is translated into an actionable command to send over to the bot; further details about the command are given later in this section

e. The listener then invokes the serial communication code to send the command over Xbee

f. The bot receives the message over the Xbee module installed on it

g. The embedded C code decides what to do with the command - either move the bot or move one of the two servo motors we have used on the robotic arm

h. One servo motor is used for up-down motion of the arm; the other for the gripper

Commands: The command that is sent from the laptop via serial communication to the bot is a simple 8-bit integer, and so lies between 0 (0x00) and 255 (0xFF). This range of values is segregated as follows:

- **0 - 9:** Reserved action commands; 2, 4, 6 and 8 are used for bot motion in the four directions
- **10 - 90:** Used for up-down robotic arm motion; the range of values in this segment is 80, which corresponds to the range of values the servo motor can take; in the C code on the bot, we simply add a constant to the value received to get it in the proper range

- **91 - 160:** Used for gripper motion; again the values are mapped linearly over to a suitable range in the C code

Important note: Do not send ASCII values. For example, to move the bot forward, writing '8' on the output stream will send the ASCII value of '8' and this won't work. Instead the 8-bit number 0x08 must be written out onto the output stream.

Figure 3: Firebird V bot with robotic arm mounted on it

Figure 4: Xbee Wireless Module

5 Working of System and Test Results

In this section we describe how a user can control the bot:

(a) **Movement of bot:** Movement of the bot is controlled using the keys A, W, S and D for left, forward, backward and right respectively. This is done using the Java Swing Interface [8]. This interface also has a video feed from the Android device mounted on the bot.

(b) **Movement of arm:** For raising or lowering the arm, the user orients his/her palm accordingly in front of the screen. A horizontal palm indicates a completely lowered arm, a vertical palm denotes a raised arm, and intermediate orientations are mapped to intermediate orientations of the robotic arm.

(c) **Gripper motion:** The gripper is controlled by pinching the forefinger and the thumb. Bringing them closer will close the gripper, taking them apart will open the gripper.

(d) **Locking:** To prevent the gripper from opening while moving an object, locking is needed. This is done using the left hand. To lock, bring your clenched left fist into the field of view of the Leap Motion sensor, with the back of your palm facing the screen. With the left hand in view, the gripper motion is disabled. Once the left hand goes out of view, the gripper can be controlled again.

Testing strategy and results:

No.	Functional Requirement	Testing Strategy	Results
1.	Motion of the Bot	Send control signals for forward, backward, left and right	Bot moved in appropriate direction
2.	Video feed from the Bot	Start streaming server on Android phone mounted on bot and try to connect to the server	Video streaming successfully started on remote machine
3.	Recognition of Gestures via Leap Motion	Generate a valid gesture and check output produced by the Leap Motion API. If it matches with the definition, then this test has been passed.	We tested this by printing out various parameters like orientation of palm normal, distance between fingers, etc.
5.	Basic Movement of Robotic Arm	Make the robotic arm perform the required actions such as moving up or down and grabbing objects with its gripper from a program loaded on the FBV.	This was tested by setting the servo motor orientation to certain values. This is also how we determined the limits of the arm and gripper orientation.
6.	Response of the bot to gestures from the user	Combining the previous two tests, generate valid gestures and check that the arm produces expected motion. That is, if it can move up or down and grab objects as per defined gestures, it is said to have passed the test. Test to be repeated with a variety of objects such as plastic objects, tomatoes, amongst others.	We attempted this test on several objects like empty paper cups, thermocol blocks and a rubber ball. In each case, we were able to pick, move and place the object.

6 Discussion of System

In this section, we describe the system in relation to the plan in the Software Requirements Specification (SRS).

(a) **Components that worked as planned:**

- Hand gesture and motion capture by the Leap Motion sensor
- The Java module that uses the Leap Motion API to decode hand gestures into signals for the bot
- Movement of the bot using keyboard controls
- The C code embedded on the FireBird to interpret signals as commands for motion of the bot or the robotic arm

(b) **Additions made to original plan:** We implemented an additional feature - locking of the gripper. After gripping an object with the gripper, due to noise in the hand motion capture, the gripper may be loosened. To avoid unintentionally releasing the object, we introduced another gesture (with the left hand) to lock the gripper in place. The gripper will not move unless unlocked with the corresponding gesture.

(c) **Changes in the orignal plan:**

- Due to problems in configuration of the bluetooth module, we had to abandon our original plan of using an Android phone for remote communication. Instead, we used a Xbee module to directly send commands from the laptop to the Firebird V bot.
- Due to problems with the Android SDK, we could not re-use code from the previous project that implemented video streaming. Instead, we used an Android application called 'IP Webcam' [6] to capture frames from the phone and retrieve them over the internet at the laptop for the user.

7 Future Work

The following components of the system can be extended and/or modified, to improve upon the work done so far:

(a) **Extensions to code:** The Java code that leverages the Leap Motion API to map hand gestures to signals to the bot is a reusable component that can be easily modified or extended as needed. The embedded C code on the bot is self-contained and is reusable. Improvements can be made in the hand gesture detection code to reduce the effect of noisy data received from the Leap Motion sensor. Likewise, the embedded C code on the Firebird V robot can be modified to enforce smooth arm motion.

(b) **Better wireless communication:** Using a bluetooth module or a WiFi card to send signals to the bot remotely is an essential improvement. Remote control, currently achieved through a Xbee wireless module is constrained by range. A WiFi card set up on the bot used to directly communicate with the main control center (the java application that decodes hand gestures) is the best option. Another alternative is to establish a bluetooth connection from the laptop to an android phone mounted on the bot that also serves the purpose of video streaming. Connection is established over WiFi to the android device which communicates with the Firebird V using a bluetooth module connected to the bot.

(c) **Complex arm motion:** The 4-axis robotic arm used in our project has 5 servo motors in all, out of which we have used only 2. Incorporating other available servo motors on the robotic arms for better control is a possible extension to

our project. While two servo motors suffice to cover basic movements (aided by movement of the FireBird), incorporating other degrees of freedom will lend a smoother and more human-like texture to the motion of the robotic arm, which will allow the arm to grasp and handle objects better.

(d) **Better gripper:** Use of a gripper with pressure sensors would greatly help improve the performance. If the gripper is loose, the object is not locked in the gripper and may fall off during transportation. On the other hand, if the object is gripped too tightly, the servo in the gripper is liable to damage. The problem is compounded by the fact that it is difficult to identify how tightly the gripper has to be shut on the basis of the streamed video. A gripper with pressure sensors would adjust automatically to each object depending on the size and this is a desirable property to have.

(e) **Depth map instead of video feed:** Instead of a simple 2D video feed, we can use a 3-D camera that will give the user a depth map of the bot's surroundings. A 2D video feed may cause problems because the user has no sense of depth, and this causes problems when the user has to grip and lift an object.

(f) **Power:** The battery on the Firebird V bot will provide power for only 1 - 1.5 hrs on a full charge. Alternative sources of power are needed if the bot is to be employed and controlled remotely.

8 Conclusion

In this document, we have described a pick-and-place robot with an intuitive hand gesture interface. We have detailed the problem that we set out to solve. In section 3, we have listed the requirements of such a system, and demonstrated our approach to solving the problem. We have described the components of our system in detail, and also the interaction between them. Section 5 contains the results obtained from our testing strategy. In section 6, we have discussed the work done in relation to the Software Requirements Specification. We then concluded with a list of possible improvements and extensions to our project.

9 References

[1] AVR Studio 4,
http://www.atmel.in/tools/STUDIOARCHIVE.aspx

[2] AVR Bootloader,
http://www.nex-robotics.com/index.php?option=com_content&view=article&id=115&Itemid=83

[3] Leap Motion Website,
http://www.leapmotion.com

[4] Leap Motion SDK and Device Drivers,
https://developer.leapmotion.com/downloads

[5] Java Simple Serial Connector,
https://code.google.com/p/java-simple-serial-connector

[6] IP Webcam,
Android app for video streaming over a network,
https://play.google.com/store/apps/details?id=com.pas.webcam

[7] Remote Robot Controller over TCP/IP,
Project by Saif Hasan et al),
http://eyantra.org/home/projectswiki/item/187remoterobotcontrollerusingtcpipconnection

[8] Java Interface for Leap Motion device,
Repo on BitBucket,
https://bitbucket.org/cs308projectgroup/handy